JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 40, 89-105 (2024)
DOI: 10.6688/JISE.202401_40(1).0006

A Proposal of JYAGUCHI Computing Platform to Realize ClouEdge (Cloud-Edge) and Serverless Architecture*

BISHNU PRASAD GAUTAM[1], AMIT BATAJOO[2,+] AND NORIO SHIRATORI[3]
[1]*Department of Economic Informatics*
Kanazawa Gakuin University
Kanazawa, Ishikawa, 920-1392 Japan
E-mail: gautam@kanazawa-gu.ac.jp
[2]*Member, IEEE*
E-mail: amitbatajoo@gmail.com
[3]*Research and Development Initiative*
Chuo University
Tokyo, 192-0393 Japan
E-mail: norio.shiratori.e8@tohoku.ac.jp

Cloud computing is a well-studied topic, but it does face some challenges, such as a lack of dynamic service deployment and selection features that can support both edge and cloud computing environments. When end users and service providers have no choice but to deploy their services in the cloud, performance issues for latency constrained cloud applications may arise. To address such challenges, we proposed ClouEdge (*i.e.*, Cloud-Edge), an optimized cloud computing infrastructure built on a JYAGUCHI computing framework that supports categorization of the services before the deployment is executed. It ensures secure and dynamic service delivery and provides deployment options to the users. End users can decide the location of deployment either in cloud or edge as per the sizes of the services that work in both cloud and edge computing environments. The proposed architecture is a novel system built on top of a public JYAGUCHI platform that can dynamically optimize and deploy services in the cloud or on-premises based upon user intention and requirement. We argue that in order to attain the appropriate level of quality, additional processes and operation layers are needed in addition to just delivering cloud resources to the edge. This architecture also performed well for latency-constrained cloud applications. We evaluated our system by testing the resource efficiency of each component. Applications and services that are latency-sensitive, will benefit from our proposed architecture such as various distributed IoT and AI services, which do not fit well with the new concepts and platforms available today.

Keywords: JYAGUCHI, cloud computing, edge computing, service delivery, serverless architecture

1. INTRODUCTION

The development of cloud services and their application in ICT industries is accelerating in order to meet the needs and challenges of end users. Cloud computing has been introduced in different areas including industrial IoT and mass manufacturing to transform and optimize manufacturing processes [1]. However, cloud-based systems are facing a number of challenges. For example, micro services built in cloud systems, are replacing many legacy systems that are difficult for end users to update and maintain. Similarly,

Received November 1, 2022; revised December 14, 2022; accepted April 26, 2023.
Communicated by Xiaohong Jiang.
* This work was supported in part by the Japan Society for the Promotion of Science (JSPS) under Grant 19H04-101 and Grant 18K11273, the Cooperative Research Project Program of the Research Institute of Electrical Communication, Tohoku University, Japan.

security challenges [2, 3] arise during service delivery from the service provider to the service consumer in cloud computing technologies which are primarily based on service-client architecture. It is a difficult issue to deliver service securely to the target end user due to the growth of services and its complex cloud and edge architecture [4, 5]. To address such issues, the concept of JYAGUCHI [6, 7] was initially introduced to demonstrate how this can be done and deployed securely to the client as a service.

The concept of software delivery (as opposed to Software as a Service) originated with the JYAGUCHI platform, which literally means "tap" in Japanese. JYAGUCHI's philosophy is that the user should be able to consume the service and control the frequency and duration of service usage, among other things, just as a tap can control the intensity or rate of flow of water.

JYAGUCHI architecture enables users or service consumers to consume services on a pay per-use basis. Furthermore, JYAGUCHI was built on the idea of leasing an entire service item or application from a service provider rather than completely owning that software by installing and licensing software. From the perspective of a novice user, software installation, license management, updating, and upgrading are always being a critical issue. However, providing those features, as implemented in JYAGUCHI, reduces the users' management and learning costs. This type of concept is also used in SaaS-based applications and more recently in serverless architecture. JYAGUCHI is more than just a SaaS (Software as a Service) platform; it is also a service development platform that can be utilized to model, develop, and export the services.

In JYAGUCHI platform, services are classified into Micro, Macro and Mega services according to the varieties of features implemented in services. These features are size, prices, end-to-end delay operations, involved techniques and configurations, server available premises, and users' interest in the services. The core and primary feature which is taken as a deciding factor is the size of the service. While the size of the service exceeds the predefined threshold, it is considered as a mega service in JYAGUCHI and recommended to keep it within the edge.

Fog computing is another computing method that has similar features with JYAGUCHI. This new paradigm, which we referred to as the "fog" in this work, enhances the cloud

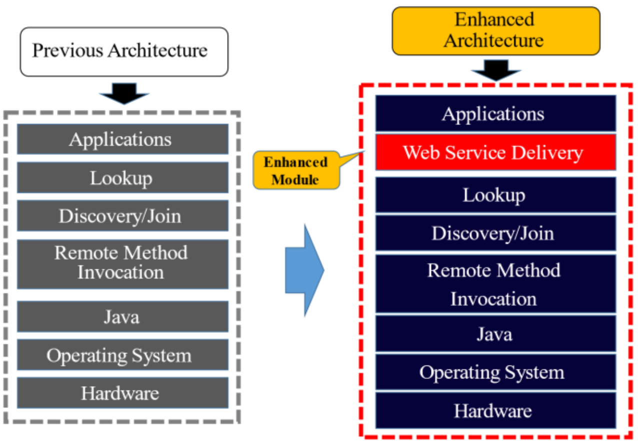

Fig. 1. Comparison of JYAGUCHI architecture scenario between previous and current enhanced designs.

computing paradigm beyond the plane of the data center to the clusters of end-user devices. Whereas Cloud computing has consolidated the computer infrastructure into huge data centers by primarily shifting computing resources from the user plane to the data center plane. Fog computing, in contrast, decentralizes resources from cloud data centers to users or the network's edge, enabling a new generation of services and applications with greater potential. Fog computing, in particular, is a paradigm for computing that moves data processing, service utilization, networking storage, and analytics closer to the applications and devices that are used by consumers [8, 9].

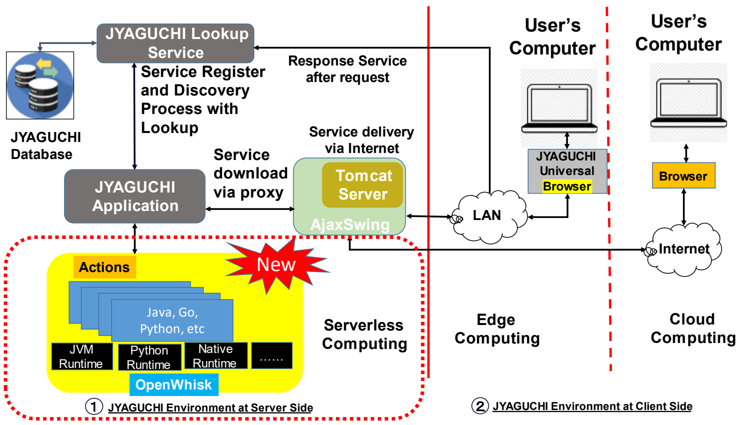

Fig. 2. JYAGUCHI ClouEdge (Cloud-Edge) and serverless environment.

The main goal of Cloud computing is to leverage the Internet and provide on demand access to fundamental computing resources. For instance, cloud users can utilize and share processing power, storage space, bandwidth, memory, applications, and software in several ways. In response to their usage, cloud providers charge the users as per their consumption. This sort of business concept has been derived from the concept of utility business and thus cloud computing sometimes refers to utility computing too. In this way, users are not required to set up or to buy hardware by themselves as it used to be traditionally. It has brought a huge paradigm shift in the market. However, it has not addressed all issues raised in the user front.

Regardless of its supremacy in terms of providing resources to the end users, it has a number of issues. Cloud computing has arisen with new data security challenges [2-4, 10, 11]. Existing data protection mechanisms such as encryption have failed in preventing data theft attacks, especially those perpetrated by an insider to the cloud provider.

1.1 Contributions of This Research Project

We believe that this work has made the following contributions:

(A) It Proposes and Demonstrates a Novel Approach in Cloud-Edge Computing Architecture: This article proposes a method for realizing a cloud-edge infrastructure that is more effective, scalable, and affordable, representing a novel approach in the field of cloud-edge computing architecture. Moreover, the JYAGUCHI architecture categorizes services according to their size and granularity. Each service in the JYA-

GUCHI architecture controls its own resources and interacts with other services through APIs, utilizing a decentralized approach to manage the services. By eliminating the need for a central controller, this method reduces the risk of single points of failure and enhances the system's fault tolerance.

(B) **It Considers Sustainable Planet Concept in Design:** JYAGUCHI is aware of carbon footprint in its designing philosophy. This principle led us to optimizes resource utilization by dynamically allocating resources to users based on their usage patterns. Users are recommended to utilize the appropriate services as per their underlying computing resources. For example, if the user has sufficient memory, storage and network bandwidth, they can use the mega and macro services via internet. Otherwise, users with limited computing resources are prompted to utilize the service locally. This approach reduces computing resource waste and improves the overall efficiency of the cloud infrastructure and contribute to reduce carbon emissions in digital products and solutions.

(C) **Improved Scalability:** The legacy JYAGUCHI's architecture has limitations due to its dependency in JINI based technology. We improved it make it highly scalable, allowing for the easy addition of new microservices as demand grows. This scalability ensures that the system can handle increasing workloads without sacrificing performance.

(D) **Cost-Effective:** To improve carbon emission, we employ the granularity-based deployment boundary. This means while the service is too large, we recommend deploying it locally. Furthermore, by utilizing pay per use basis technology and a decentralized management approach, JYAGUCHI reduces the cost of deploying and maintaining a cloud infrastructure. This cost-effectiveness makes it an attractive option for small to medium-sized businesses that want to leverage the benefits of cloud computing without incurring high costs.

2. RESEARCH CHALLENGES

The distributed application architecture like JYAGUCHI must consider lots of architectural elements, components, connectors and other parameters of the system that directly or indirectly affect the realization of the system. These elements are required to be analyzed and be designed to depict the solution before implementation leading to the best design decision in order to reduce the total cost of the system. In this paper, we figured out the following research challenges which are significant to address.

2.1 Secured Service Delivery

Security is always a challenging issue in the cloud while providing different types of services to users. It also may reveal information which adds to security issues and risks of cloud computing systems. Most security issues on the Internet are common to existing computer security problems in communication and the downloading of software. As cloud provides a Software as a Service (SaaS), a comprehensive solution offering the entire package from infrastructure to application, service or package delivery securely is still a challenging issue. In order to address these security issues, it is important to identify a common security problem not only in communication protocols but also while downloading and uploading events happened between the user and servers. Furthermore, these issues apply to cloud computing, and thus must be addressed in order to ensure the security and privacy of data in the cloud. To address this challenge, a robust access control mechanism must be implemented that can handle during service delivery in the cloud system. Furthermore, this mechanism should ensure that only authorized users and applications are granted access to the cloud resources, and that any unauthorized access attempts are immediately detected and prevented.

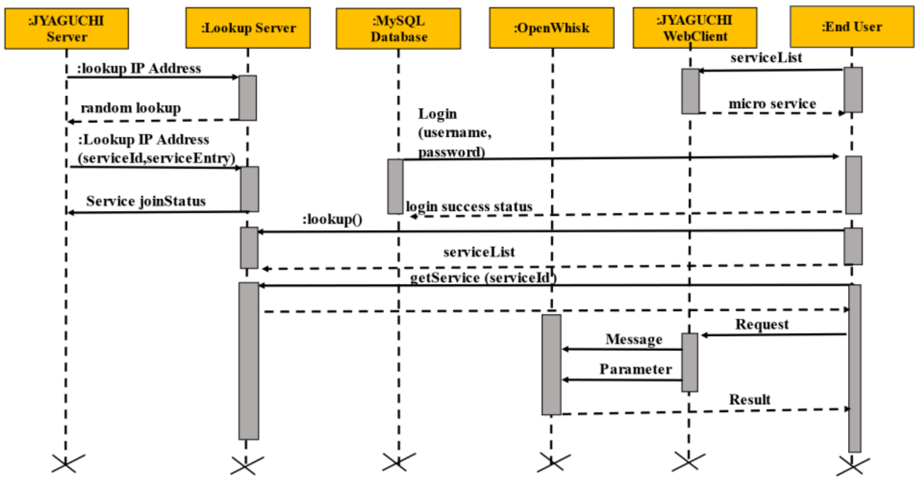

Fig. 3. JYAGUCHI sequence diagram enhanced.

2.2 Limitation in Service Lookup in Legacy System

The lookup service in previous JYAGUCHI serves as a central repository of services. Entries in the lookup service are Java objects, which can be downloaded as local proxies to the service that registered with the lookup service [12].

2.3 Dynamic Service Delivery

The network protocol that is used to communicate between a discovering entity and an instance of the discovery request service is assumed to be unreliable and connectionless, and to provide unordered delivery of packets. In an environment that makes use of IP multicast or a similar protocol, the joining entity should restrict the scope of the multicasts it makes by setting the time-to-live (TTL) field of outgoing packets appropriately [12, 13]. This maps naturally onto both IP multicast and local-area IP broadcast but should work equally well with connection-oriented reliable multicast protocols.

3. PROPOSED SOLUTION

To address the above challenges, we proposed the enhanced JYAGUCHI architecture which incorporates the essence of architectural styles adopted in the distributed applications and understand its inherent problem. The architectural differences between the previous JYAGUCHI and enhanced JYAGUCHI is portrayed in Fig. 1. The original architecture of JYAGUCHI [15] we started to analyze the widely used architectural style in the file of distributed applications and further deepen our understandings of the internal architecture of those applications. In addition, we have developed some services which were developed emphasizing on the scalability of components and the reduction of dependencies among them. Furthermore, it maintains the principle of encapsulating legacy systems by providing the simple methodology of interfacing the underlying software components and the way of enhancing them to be well defined services [3].

To address the enhanced proposed solution, there is no longer any need for distributed Java applications to be installed directly on the user's computer anymore, as with proposed in Section 4 *i.e.*, Enhanced JYAGUCHI architecture it is now accessible via Internet by entering the correct URL address.

4. ENHANCED JYAGUCHI ARCHITECTURE

Distributed application varies in granularity and infrastructure. Traditionally, distributed applications were two-tiered, three-tiered or multi-tiered in their architecture which collectively makes a single system. This notion of single system has evolved to the creation of from tire-based system to a huge virtualized system such as grid system [16, 17]. We experienced that the emergence of cloud computing as a new platform for enterprise. From its very inception, JYAGUCHI service development model utilizes the legacy computing infrastructure thereby creating a cluster of possible hardware that can participate in the federations of JYAGUCHI services. The hardware and services that participate in JYAGUCHI services are capable of addressing the problem in a co-operative manner. The notion of co-operation has been executed by utilizing the concept of SOA in which the underlying computing infrastructure or underlying middleware are encapsulated and the detailed of which are not required while providing the services to the end user. In fact, JYAGUCHI services can be built in a number of multiple technologies and protocols [18-20]. Though there are the different architectures underlying, JYAGUCHI service models can produce similar characteristics of web service and can be used together with other web service like technologies too. This ability is referred to as co-operative computing infrastructure. Fig. 3 shows the relation between each device and the underlying software components that cooperate while developing, deploying and using of JYAGUCHI service. In particular usage scenario, JYAGUCHI client send request to the lookup server, this server provides the proxy required to the client and with the help of this proxy, client will be able to interact and can download the remaining codes from web server. In this way, a solution is achieved. In order to scale out the co-operative infrastructure, the underlying hardware federation can be increased by virtualization. We are also exploring the ways to build the services by using encapsulation service modularization approach [18]. We have developed a complete package of middleware by integrating different kinds of underlying technologies.

We agree that JYAGUCHI platform supports the development of distributed objects. The main requirement of distributed object is its ability to create, invoke and deliver the objects in a remote host while providing the environment as if they were invoked in local infrastructure. These sorts of remote object invocation have been implemented in COM, COBRA and RMI and many other technologies until a few years ago. JYAGUCHI employs a similar kind of concept that invokes the total bundle of service executed on a remote server. We named this invocation model as RSI. The underlying protocol to call the remote service is JERI and JRMP [12, 21, 22]. Recently, a different type of invocation model is often utilized in web service technology such as WSIF [13, 23, 24]. Web service technologies have passed different stages of evolution in terms of utilizing underlying message passing protocols such as SOAP and REST.

These technologies, must of the time, utilize XML data format to send and receive the message. Messages can also be passed in JSON format too. However, service call in web service technology and JYAGUCHI is different. Most of the web service-related technologies utilizes message passing technique whereas JYAGUCHI utilizes RSI at which parameters are passed as the reference of java object. We did not utilize message passing rather we utilized service calling approach in order to reduce the overhead occurs in message passing. In message passing, it has to copy the existing arguments and append it to the new portion of the message resulting to a large size of message.

JYAGUCHI emphasizes loose coupling of the components thereby reducing the dependencies of the components participating in the foundation for the architecture. The overall architectural style presented by JYAGUCHI never tries to replace the prevalent architecture but tries to leverage and show the guideline for the next generation applications by utilizing hybrid architecture style [2]. The post notification and dynamic delivery of the service omit the requirement of complex procedure of software installation for the client.

Particularly, these services are coded as Java objects and are wrapped with JYAGUCHI service and the whole service is remarshaled in the user device. In the following sections, we describe the total scenarios and implementation approach of JYAGUCHI, service wrapping scenarios and the concept of service granularity. In our new approach in the Fig. 1, we have implemented the new platform for secure deliver web server which simply run JYAGUCHI application and is able to be delivered and accessible via Internet, by just entering the correct URL address. We also ensure stable performance and regular delivery of JYAGUCHI application via internet. As the web interface became the primary platform for software distribution in cloud. The delivery of JYAGUCHI services has become platform and device independent.

5. IMPLEMENTATION

The concept of modularization is to separate the concern and context from one piece of program another so as to minimize the effect that changes in one module may have on other modules. Separating the unrelated concerns from the modules has a great advantage of reducing interdependencies of modules so as to minimize the coupling between the server and client program. While there is a maximum coupling between server and client program, a small change in server program needs to be informed to the user. While this sort of software cohesion and coupling issues can be addressed in service modularization [25, 26] and tenancy classification [27]. Our approach utilizes loose coupling and service categorization, as described below, which we believe to be both novel and practical for cloud-edge infrastructure:

5.1 Micro Service

Micro services (Fig. 4) in JYAGUCHI are the services which can be downloaded over a network and these services can be exported as a complete software package. To utilize this service, JYAGUCHI client does not even require knowing about the interface. However, JYAGUCHI client must possess a universal browser, where we have implemented a universal interface that can be used to call any JYAGUCHI mini service. These services are implemented for the users who have low internet bandwidth.

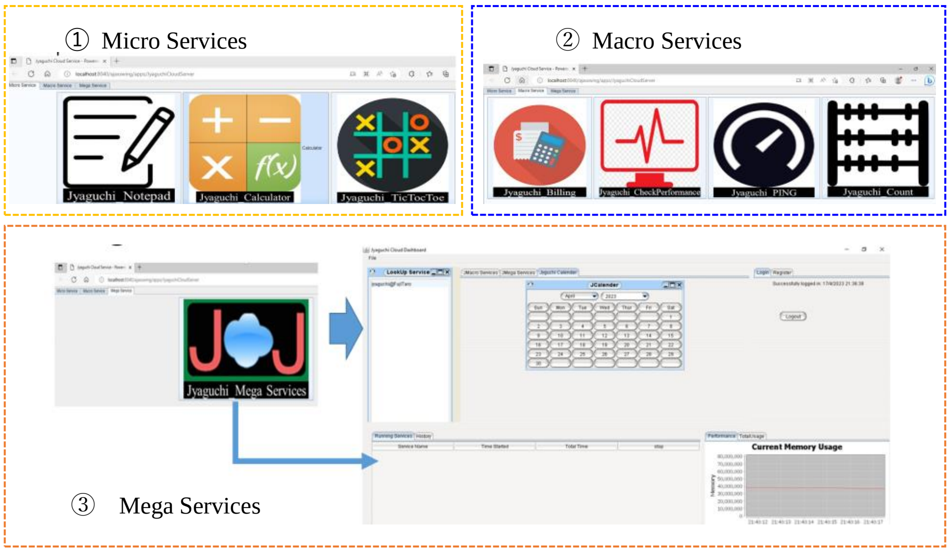

Fig. 4. JYAGUCHI services (micro, macro and mega) scenarios.

5.2 Macro Service

Macro services (Fig. 4) are the services which can either be downloaded or can be accessed via web browser. As the size of service becomes larger than micro service, we categorized these kinds of services to macro service and put the option to the client to choose whether he or she wants to download the entire service or can access it by using a web browser as like other web services.

5.3 Mega Service

Mega services (Fig. 4) in JYAGUCHI are those services whose granularity is extremely larger than macro services and are not feasible to serialize as a complete software that can be done for micro and mega services. Option is also omitted, and clients can only use this kind of service like web services by using a web browser. In order to ensure the access of service at different network mode as in Fig. 2 we have proposed the solution as follow.

5.4 JYAGUCHI Service at Public Network (Within Cloud)

Using the web browser, users can access the JYAGUCHI client system. The network area with the public or outside the organization or company will allow the users to access only Micro service. The Mega & Macro services are disabled, and users are not allowed to access them.

5.5 Security in JYAGUCHI

The design of the security model for JYAGUCHI is built on the twin notions of a principal and an access control list. JYAGUCHI services are accessed which are generally traced back to a particular user of the system. Services themselves may request access to other services based on the identity of the object that implements the service. Whether access to a service is allowed depends on the contents of an access control list that is associated with the object [14, 28].

The solution provided by cloud infrastructures raises security concerns regarding data privacy, as data stored in the cloud may not be protected with the same level of security as data stored in an end-user's own infrastructure. Third-party data centers cannot guarantee data and privacy protection [2]. In JYAGUCHI, data is not retained on the server-side, but rather on the client-side to give end-users control over their data. This solution increases trust in the system, as no data is transferred to the server-side, except for data related to user authentication. Once the user successfully logs in, the client can send a request to search the service registry in the network. To find the registry, we have tested both unicast and multicast discovery [2].

Meanwhile, in the enhanced JYAGUCHI architecture, the original concept of user authentication for the Macro and Mega services are the same in Edge environment. In addition, to ensure the Micro services are secured in a Cloud environment there is a user authentication before they access services.

In order to maintain high security for the services when user access from a Cloud environment, the information for the authentication is privately provided by the System Administrator after their request. Moreover, even when a user accesses from the Cloud environment the user's data does not retain in the server side rather, they retain in the client side so that end-user can have control over their data. This solution in fact increases the trust over the system as no data is transferred to the server side except the data related to user authentication both for Edge and Cloud environments.

6. PERFORMANCE EVALUATION

To perform performance evaluation, we began to implement JYAGUCHI applications as services in two different environment Edge & Cloud environment and tested whether we can utilize the deployment environment without having difficulties. We did not have much difficulty to implement JYAGUCHI services and expose them via lookup service and browser for end users.

6.1 Latency Evaluation

As the latency of a network is the time it takes for a data packet to be transferred from its source to the destination. In Fig. 5-A, we can observe that after the server starts, delays in transmission are small, it's referred to as a low-latency network. This is the latency for downloading the Micro service.

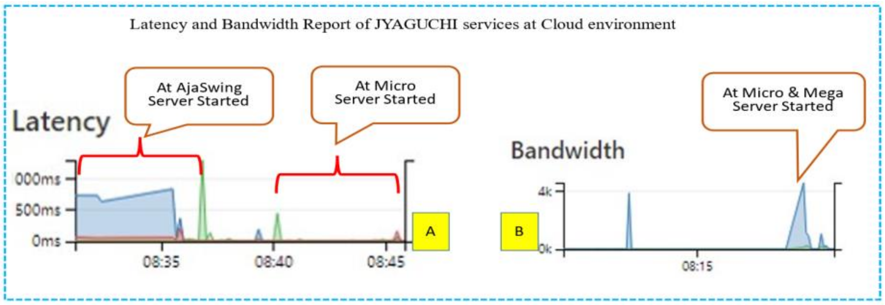

Fig. 5. Latency and bandwidth report of JYAGUCHI services at Cloud environment.

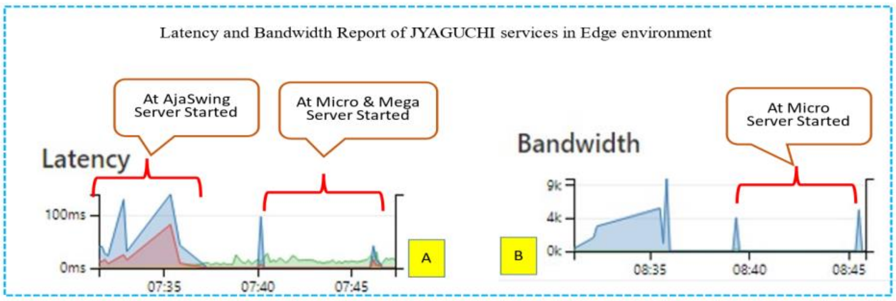

Fig. 6. Latency and bandwidth report of JYAGUCHI services in Edge environment.

On the other hand, in Fig. 6-A we can observe a larger delay when an application is started to run Mciro & Mega JYAGUCHI service. This high latency is due to Mega service download from the JINI server to the user's computer. But after download is completed to the local PC of user's the latency gradually decreases.

6.2 Bandwidth Evaluation

The JYAGUCHI services access at the socket or stream level. Below this level, the data is handled on the network, using the appropriate protocol. Depending on the Internet speed of the user the bandwidth may vary. The Internet download and upload speed was 38.36 Mbps and 43.14 Mbps respectively during the evaluation process in our working environment. We can observe in Fig. 5-B as a bandwidth Report of JYAGUCHI services at Cloud environment that with the minimum internet speed the Micro service is accessible to the user's computer via the Internet.

In Fig. 6-B, to access the Mega service via the Internet there is a little higher peak of bandwidth. This is because the application downloads in the user's computer via Internet developed in Java swing are difficult to carry from server to end users' due to limitations of network and platform that end users can access simply with no expert technical knowledge.

Here in JYAGUCHI architecture we are proposing a dynamic ability where we can deliver Java swing applications to end users simply using a Web browser. By doing this we can re-use the Java swing code in the development of valuable services without wasting time and money for learning new languages and changing the development and deployment environment in academic or enterprise environments.

7. SUSTAINABILITY ASPECT IN JAYGUCHI

7.1 Aspect of Sustainable Layer

The quest for sustainable computing is one of the most challenging issues in cloud computing today. To address this challenge and contribute to a more sustainable future, JYAGUCHI is taking a multi-faceted approach that considers several key modules and aspects. These include the implementation of a sustainable layer consisting of a power measurement module, dynamic resource allocation module, energy monitoring and reporting module, and energy efficiency module. Additionally, JYAGUCHI leverages its legacy code, particularly old Java Swing codes, to minimize energy consumption and reduce the carbon footprint of its development process. This approach not only saves time and resources but also helps to minimize the overall environmental impact of the software system. Nonetheless, it is important to note that legacy code may not be as efficient or effective as newer code, hence it is recommended to measure energy consumption using energy measurement functions or third-party libraries like jRAPL, PowerAPI, and EnergyMon.

7.2 Aspect of Web Enabled Swing Component

JYAGUCHI was developed entirely on Java programming language where many of its applications were built on Swing components initially. Java is a widely used programming language across various computing platforms including cloud infrastructure. However, with the emergence of new programming languages such as Python, Go, C#, R , and Rust many organizations and IT companies are now adapting to these newer options. These newer programming languages are rises in popularity for cloud-native applications and computing environments. As a result, the vast majority of apps created using Java Swing are now limited for personal or academic research and education. This is primarily because Java Swing applications are difficult to move from the server to the end user due to network and platform accessibility restrictions, which often requires advanced knowledge of distributed computing and security. Furthermore, Swing components provided in Java computing platforms have GUI (Graphical User Interface) elements and thus may consume more energy while transferring them via internet while the granularity of Swing components are larger. To tackle energy consumption issue, a granularity-based categorization method was proposed in latest JYAGUCHI architecture which we believe substantially reduces carbon emissions and promote to achieve the sustainable earth. Additionally, the JYAGUCHI architecture provides a dynamic method for providing end users with simple web browser access to Java Swing applications. This web interface provides options to use the platform either locally, or in edge or cloud. In addition to increasing accessibility, this method also makes it easier for end consumers to utilize, regardless of their level of technical proficiency. By utilizing this architecture, users can access Java Swing apps without needing to go through any complicated technical training or installation processes, making it possible for a larger audience to take use of the Java Swing-developed applications. In this way, the updated JYAGUCHI architecture engage users less time in computing resource and thus contributes to reducing carbon emission.

7.3 Aspect of Green Energy Usage

Another important aspect to consider for a sustainable layer in JYAGUCHI's cloud system is the use of renewable energy sources. By leveraging renewable energy sources such as solar, wind, or hydroelectric power, JYAGUCHI can reduce its dependence on non-renewable energy sources and contribute to a more sustainable computing ecosystem. Additionally, optimizing the data center's cooling systems and using energy-efficient hardware can also help in reducing energy consumption and the overall carbon footprint. By implementing these sustainable practices, JYAGUCHI can demonstrate its commitment to sustainable computing and contribute to a more environmentally friendly technology industry. However, considering all of these aspects in the architecture of JYAGUCHI alike cloud system and the implementation of a sustainable layer is still a task for future work.

8. COMPARISON BETWEEN DISTRIBUTED SERVICES

In this section, we conducted a study and comparison of similar distributed services to gain insights into the extension and limitations of the JYAGUCHI architecture. Comparing all prevalent sets of relevant features such as Ease of Use (EoU), Ease of Development (EoD), performance, cost, security, openness, and sustainability would be a daunting task, and citing all related works would be too extensive. Therefore, we only compared JYAGUCHI with selected works, including Apache River, WebSwing, gRPC, and Vaadin, which are related to the seven quality attributes. We evaluated each architecture using a set of symbols: a double circle represents "very good," a circle represents "good," and a triangle represents "fair," as shown in Table 1.

Table 1. Comparison between distributed services.

Quality Attribute / Distributed Service	Ease of Use (EoU)	Ease of Deployment (EoD)	Security	Performance	Openness	Sustainability	Cost
JYAGUCHI	◎	◎	○	○	◎	◎	◎
Apache River	▲	▲	▲	▲	◎	▲	◎
WebSwing	◎	▲	○	○	▲	◎	▲
gRPC	○	○	◎	◎*	◎	▲	◎
Vaadin	○	○	○	○	○	◎	▲

◎ Very Good ○ Good ▲ Fair

* In terms of gRPC performance indicator is marked as very good; however, gRPC does not transport GUI object (Swing) as service to the end users but sends only the data. Thus, the performance of gRPC outperformed the other methods which may seem unfair but we do not count this specific features in our comparison.

The objective of comparing in this way is to clarify the definition, use, and emphasis those quality attributes presented in Table 1. Such subjective comparison will help us to determine the extension and limitation of JYAGUCHI architecture, which will help us improved design in future. In our evaluation based on subjective analysis, we found that

JYAGUCHI architecture is able to provide all the features required for distributed computing and are capable of service delivery in distributed environment; however, these systems have no architectural guarantee of EoU, EoD, sustainability, security and openness feature as proposed in JYAGUCHI architecture.

Furthermore, we definitely think that this kind of comparison will help researchers while choosing the appropriate cloud-edge architecture in their research project.

9. INTEGRATION OF SERVERLESS APPROACH IN JYAGUCHI

To test the serverless approach, we designed services in the upgraded JYAGUCHI for particular applications. JYAGUCHI is a fully functional distributed architecture that also supports serverless features. In this study, we have created a cloud-edge and integrated this platform with a serverless platform to improve the capability of wide-area service distribution without the restriction of server management. Many IT industries such as Amazon AWS, Microsoft Azure, and IBM Cloud, have already begun to offer their clients a new, paid platform called serverless.

Table 2. Details of server specification.

Operation System	RAM	CPU	Disk
Ubuntu 18.04 (LTS) × 64	2 GB	1	50 GB

Here, we integrated JYAGUCHI with the OpenWhisk (Fig. 7) engine. Table 2 shows the details of server specification during implementation. In order to implement and trigger an action, we prepare a simple JYAGUCHI micro service. While a developer at JYAGUCHI environment creates the services and pushes his service in the repository, an automated message triggers to the JYAGUCHI Environment at client side. For the testing purpose, we have used Slack, a messaging app that connects people to the information. Inside OpenWhisk engine the generated action as follows,

① Action create: This will create a new action
② Action invoke: This will run an action

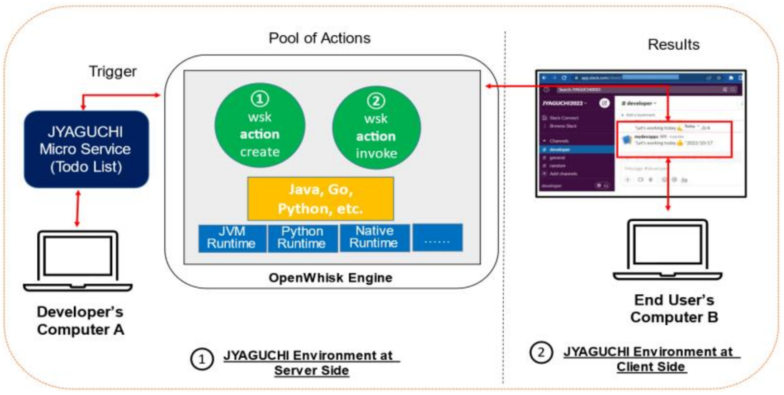

Fig. 7. Integration of serverless architecture in JYAGUCHI.

In this proposed architecture we are able to successfully implement serverless service with JYAGYUCHI service.

10. CONCLUSION AND FUTURE WORKS

In this study, we architected JYAGUCHI and successfully demonstrated a service-oriented development approach that enables users to create, publish, and use services in accordance with their needs and security constraints. For instance, consumers are advised to use the edge computing platform offered by JYAGUCHI if the client's communication infrastructure is unreliable and bandwidth sensitive. In this case, mostly the mega and macro services are developed and delivered to the end users within the edge environment. However, if the services are relatively small and may not consume high bandwidth, those services can be located in the cloud side either in server-client model or in Serverless format. In this research, we enhanced our previous JYAGUCHI platform which was based on JINI [12, 21, 29] technology by proposing a new architectural component that augments the services as per the granularity of the applications and also ensure the access control for secured service delivery. We evaluated our system by testing the performance of each component in terms of resource consumption, latency and network bandwidth metrics within and outside the edge. Furthermore, we integrated the Serverless approach into the architecture. In our previous architecture, distributed services were not able to pass via the internet due to the restriction in NAT. However, this constraint is solved by designing a NAT traversal module in the architecture. In the current architecture, this module is integrated with AjaxSwing and the services can be accessed by using a web browser.

We have also incorporated sustainability considerations in our design. We are aware that data centers, cloud computing infrastructures, and digital services consume a significant amount of energy and produce carbon emissions. We strongly believe that our innovation and solution should adopt a sustainable approach and demonstrate a commitment to creating sustainable digital services, as responsible researchers. This will help us to ensure that our work is not only effective, but also sustainable in the long term. To embrace this philosophy, JYAGUCHI has devised a sustainable layer in its architecture. However, at this stage, the sustainable layer is not fully implemented. We look forward to reporting additional findings in our future articles.

REFERENCES

1. C. Yang, S. Lan, L. Wang, W. Shen, and G. G. Q. Huang, "Big data driven edge-cloud collaboration architecture for cloud manufacturing: A software defined perspective," *IEEE Access*, Vol. 8, 2020, pp. 45938-45950.
2. A. Ometov, O. L. Molua, M. Komarov, and J. Nurmi, "A survey of security in cloud, edge, and fog computing," *Sensors*, Vol. 22, 2022, p. 927.
3. A. M. Alwakeel, "An overview of fog computing and edge computing security and privacy issues," *Sensors*, Vol. 21, 2021, p. 8226.
4. Y. Xiao, Y. Jia, C. Liu, X. Cheng, J. Yu, and W. Lv, "Edge computing security: State of the art and challenges," *Proceedings of IEEE*, Vol. 107, 2019, pp. 1608-1631.

5. A. Mulay, H. Ochiai, and H. Esaki, "IoT WebSocket connection management algorithm for early warning earthquake alert applications," in *Companion Proceedings of the 10th ACM International Conference on Utility and Cloud Computing*, 2017, pp. 189-194.
6. B. P. Gautam, K. Wasaki, and A. Batajoo, "Encapsulation of micro engineering tools in a co-operative Jyaguchi computing infrastructure," *ScieXplore International Journal of Research in Science*, Vol. 1, 2014, p. 1.
7. B. P. Gautam and D. Shrestha, "A model for the development of universal browser for proper utilization of computer resources available in service cloud over secured environment," in *Proceedings of International MultiConference of Engineers and Computer Scientists*, Vol. I, 2010, pp. 638-643.
8. B. P. Gautam, A. Batajoo, and K. Wasaki, "Fogging Jyaguchi services in Tensai Gothalo," *International Journal of Computer Trends and Technology*, Vol. 28, 2015, pp. 119-125.
9. M. Goudarzi, M. Palaniswami, and R. Buyya, "Scheduling IoT applications in edge and fog computing environments: A taxonomy and future directions," *ACM Computing Surveys*, Vol. 55, 2022, Article No. 152, pp. 1-41.
10. N. A. Sulieman, L. R. Celsi, W. Li, A. Zomaya, and M. Villari, "Edge-oriented computing: A survey on research and use cases," *Energies*, Vol. 15, 2022, p. 452.
11. G. Caiza, M. Saeteros, W. Oñate, and M. V. Garcia, "Fog computing at industrial level, architecture, latency, energy, and security: A review," *Heliyon*, Vol. 6, 2020, p. e03706.
12. M. Sun, "Jini^{TM} technology core platform specification," http://di002.edv.uniovi.es/~falvarez/core1_1.pdf, 2000.
13. M. Diarra, "Enhanced transport-layer mechanisms for MEC-assisted cellular networks," University of Cote D'azur, https://theses.hal.science/tel-03946149/file/131818_DIARRA_2022_archivage%20%282%29.pdf, 2023.
14. C. Raiciu, J. Iyengar, and O. Bonaventure, "Recent advances in reliable transport protocols," https://sigcomm.org/education/ebook/SIGCOMMeBook2013v1_chapter2.pdf, 2023.
15. B. P. Gautam, "An architectural model for legacy resource management in a Jini based service cloud over secured environment," IPSJ SIG Technical Reports, No. 2009-EIP-43, Information Processing Society of Japan, Vol. 11, 2009, pp. 55-62.
16. M. Simic, I. Prokic, J. Dedeic, G. Sladic, and B. Milosavljevic, "Towards edge computing as a service: Dynamic formation of the micro data-centers," *IEEE Access*, Vol. 9, 2021, pp. 114468-114484.
17. A. Ghosh and K. Grolinger, "Edge-cloud computing for IoT data analytics: Embedding intelligence in the edge with deep learning," *IEEE Transactions on Industrial Informatics*, Vol. 17, 2020, p. 2191-2200.
18. S. K. Shrestha, Y. Kudo, B. P. Gautam, and D. Shrestha, "Recommendation of a cloud service item based on service utilization patterns in Jyaguchi," in *Knowledge and Systems Engineering*, V. N. Huynh, T. Denoeux, D. H. Tran, A. C. Le, and S. B. Pham, eds., in *Advances in Intelligent Systems and Computing*, Vol. 245, Springer International Publishing, Cham, 2014, pp. 121-133.
19. B. P. Gautam, K. Wasaki, A. Batajoo, S. Shrestha, and S. Kazuhiko, "Multi-master replication of enhanced learning assistant system in IoT cluster," in *Proceedings of*

IEEE 30th International Conference on Advanced Information Networking and Applications, 2016, pp. 1006-1012.
20. M. S. Alam, U. D. Atmojo, J. O. Blech, and J. L. M. Lastra, "A REST and HTTP-based service architecture for industrial facilities," in *Proceedings of IEEE Conference on Industrial Cyberphysical Systems*, 2020, pp. 398-401.
21. N. Furmento, W. Lee, A. Mayer, S. Newhouse, and J. Darlington, "ICENI: An open grid service architecture implemented with Jini," in *Proceedings of ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis Conference*, 2002, p. 37.
22. J.-M. Dricot, "Development of distributed self-adaptative instrumentation networks using Jini technology," in *Proceedings of IEEE International Workshop on Virtual and Intelligent Measurement Systems*, 2001, pp. 22-27.
23. E. Gökçay, "A new multi-target compiler architecture for edge-devices and cloud management," *Gazi University Journal of Science*, Vol. 35, 2022, pp. 464-483.
24. M. Migliardi and R. Podesta, "Performance improvement in web services invocation framework," in *Proceedings of IEEE 18th International Parallel and Distributed Processing Symposium*, 2004, pp. 110-122.
25. S. Yangui, A. Goscinski, K. Drira, Z. Tari, and D. Benslimane, "Future generation of service-oriented computing systems," *Future Generation Computer Systems*, Vol. 118, 2021, pp. 252-256.
26. J. Singh, P. Singh, and S. S. Gill, "Fog computing: A taxonomy, systematic review, current trends and research challenges," *Journal of Parallel and Distributed Computing*, Vol. 157, 2021, pp. 56-85.
27. H. T. Malazi *et al.*, "Dynamic service placement in multi-access edge computing: A systematic literature review," *IEEE Access*, Vol. 10, 2022, pp. 32639-32688.
28. V. C. Hu, D. R. Kuhn, and D. F. Ferraiolo, "Access control for emerging distributed systems," *Computer*, Vol. 51, 2018, pp. 100-103.
29. J.-M. Dricot, P. de Doncker, M. Dierickx, F. Grenez, and H. Bersini, "Jini technology as a solution to develop distributed instrumentation network in engineering," in *Proceedings of International Symposium on the Convergence of IT and Communications*, 2001, pp. 16-23.

Bishnu Prasad Gautam received his master's and Ph.D. in Computer Engineering from Shinshu University. He has published over 50 papers in international journals and international conferences. He has been invited as a key speaker in several International Workshops, Conferences and Universities. He is currently working as a Professor (Full) at Kanazawa Gakuin University, and he is a member of IEEE, IPSJ and IAENG. His current research interest includes sustainable computing, distributed computing architecture, network architecture, network security, and IoT. He was a recipient of Highest Score Award (IPA), Highest Championship Award and Championship Prize (General) of ET Robotic Championship Contest in 2018, Japan. He has obtained several awards in international conferences and academic societies.

Amit Batajoo received the MS degree in Computer Science and Engineering from Shinshu University at Nagano, Japan in 2019. He is currently working in Fujitsu Limited, Japan as Software Engineer. His research interests include modeling and analysis of operational behavior, mathematical model and formal verification of software and development process, open-source software, ad-hoc network communication protocol. Also interested in the micro-service architecture, DevOps, and development of IoT system, database management system, web application and Android app. He is a member of IEEE and IEICE. Mr. Batajoo's award include Incentive Award, 62nd Intelligent Transportation Systems and Smart Community Research Workshop, Paper Title: Fogging Jyaguchi Services in Tensai Gothalo, Wakkanai, Japan, 2015.

Norio Shiratori (Life Fellow, IEEE) is currently a Professor with Chuo University, Tokyo, and also an Emeritus Professor with Tohoku University, Sendai, Japan. He has published over 15 books and over 600 refereed articles in computer science and related fields. He is a Fellow of the Japan Foundation of Engineering Societies (JFES), the Information Processing Society of Japan (IPSJ), and the Institute of Electronics, Information and Communication Engineers (IEICE). He was a recipient of the Minister of MEXT Award from the Japanese Government in 2016, the Science and Technology Award from the Ministry of Education, Culture, Sports, Science, and Technology (MEXT) in 2009, the IEICE Achievement Award in 2001, the IEICE Contribution Award in 2011, the IPSJ Contribution Award in 2008, the IEICE Honorary Member in 2012, the IPSJ Honorary Member in 2013, the IPSJ Memorial Prize Winning Paper Award in 1985, the IPSJ Best Paper Award in 1997, the IEICE Best Paper Award in 2001, the IEEE 5th SCE01 Best Paper Award in 2001, the IEEE ICPADS 2000 Best Paper Award in 2000, and the IEEE 12th ICOIN Best Paper Award. From 2009 to 2011, he was a former President of IPSJ.